复习：

见思维导图

1.RN中的常用组件——图片

**本地图片：**

<Image source={require('../src/assets/x.jpg')}/>

本地图片可以无需指定尺寸（因为导入/打包在服务器端进行），默认是原始尺寸；如果指定了width，height不会自动的修改，也必须手工赋值

**远程图片：**

<Image source={{uri:'http://t.cn/logo.png'}} style={{width:x,height:x}}/>

注意：Image的source可以赋值为“远程图片的URL地址”或者“本地的图片**数据**”——本地图片不能使用文件路径名字符串！远程图片需要异步请求，初始时图片尺寸为0；请求完成后默认尺寸仍然为0——必须手工设置远程图片的尺寸（防止页面重排）

Image组件的resizeMode决定了图片尺寸与组件尺寸不等时如何缩放，可取值：

①cover：覆盖，等比例缩放图片，保证可以完全覆盖组件的宽和高，溢出部分不显示（默认值）

②contain：包含，等比例缩放图片，保证容器可以完全容纳图片的全部内容，可能出现留白

③stretch：拉伸，不等比例缩放图片，保证容器可以完全容纳图片的全部内容，不会留白但图片会变形

④repeat：重复，等比例缩放图片，在容器中平铺图片，保证没有留白

⑤center：图片不拉伸直接放置在容器正中央

|  |
| --- |
| 重点：如何获取远程图片的原始尺寸？从而实现图片内容全部显示且无留白  Image.getSize( uri, (w, h)=>{ }, (err)=>{ } ) |
| 午间任务：RN中如何读取手机屏幕的尺寸？从而实现图片铺满屏幕宽度 |

|  |
| --- |
| 小知识：url、uri、urn三者间的关系？  Unified Resource Identifier：统一的资源识别符  Unified Resource Locator：统一的资源定位符，如http://b.com/logo.jpg  Unified Resource Naming：统一的资源命名符，如tel: 13312345676、mailto:tom@t.cn  **URI = URL + URN** |

|  |
| --- |
| 使用本地图片时的经典错误：  this.state.list = [  './src/asset1/0.jpg',  './src/assets/1.jpg'  ]  <View>  {  this.state.list.map((e, i)=>{  return <Image source={require(e)}/>  })  }  </View> |
| 错误提示： Invalid call at line xx： require(e)  错误原因： require放在循环/选择中，变为动态导入，Webpack打包时不支持动态导入！  解决方法： 把动态导入变为静态导入： |
| this.state.list = [  require('./src/asset1/0.jpg'),  require('./src/assets/1.jpg')  ]  <View>  {  this.state.list.map((e, i)=>{  return <Image source={ e }/>  })  }  </View> |

2.RN中的常用组件——活动指示器

<ActivityIndicator size="small/large" color="#f00" />

显示一个“加载中”的圆环动画图片；默认为小号，可以设置为大号；

注意：页面中最后一个“活动指示器”的color会影响前面所有指示器的颜色

3.RN中的常用组件——开关

<Switch value={ 布尔变量 } onValueChange={事件处理函数} />

属于需要进行“双向数据绑定”的受控组件，value是boolean数据

4.RN中的常用组件——触摸反馈

<TouchableOpacity>

其它组件(文字/图片等)

</TouchableOpacity>

提示：RN中提供了四个触摸反馈组件，其中TouchableOpacity是触摸后发生透明度的改变

5.RN中的常用组件——简单列表——难点&重点

高性能的列表组件——底层值挂载当前需要显示的列表项，以及上方/下方若干条即将被显示的列表项，更大范围内列的表项不进行挂载

<FlatList data={数组} renderItem={ this.\_renderItem } keyExtractor={this.\_keyExtractor} />

\_renderItem = ( obj )=>{

//obj.item：是一个数据 obj.index：是该数据的下标

return (表示一个列表项的JSX)

}

|  |
| --- |
| FlatList组件的属性：  ①data：必需的，数组类型，指定列表要渲染的数据  ②renderItem：必需的，函数类型，指定如何渲染一个列表项  ③keyExtractor：函数类型，指定如何提取每个列表项的key  ④ListHeaderComponent：函数类型，指定列表头部内容  ⑤ListFooterComponent：函数类型，指定列表尾部内容  ⑥ItemSeparatorComponent：函数类型，指定列表项间的分隔条内容  ⑦onEndReached：事件，指代滚动到了列表尾部  ⑧onEndReachedThreshold：数字，0~1之间，指定滚动到距离底部还剩多远时触发onEndReached事件 |

练习：假设有如下数据：laptopList : [

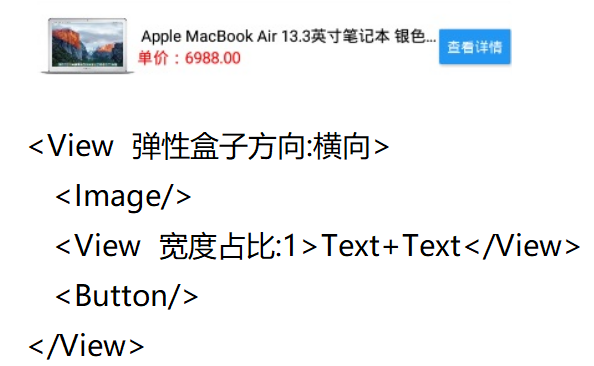
{lid: 100, title:'笔记本标题0', pic: './src/assets/0.jpg',price:500},

{lid: 101, title:'笔记本标题1', pic: './src/assets/1.jpg',price:600},

{lid: 102, title:'笔记本标题2', pic: './src/assets/2.jpg',price:700},

{lid: 103, title:'笔记本标题3', pic: './src/assets/3.jpg',price:800},

]，将这些数据渲染在一个FlatList中，其中每个列表项形如：



6.RN中路由和导航实现

提示：RN的运行不依赖于浏览器！没有window.navigator/history对象！甚至没有Page的概念！——RN中切换是Screen，不是Page。

RN官方没有提供导航组件，推荐使用第三方导航组件：**React Navigation.**

React Navigation的官网：https://reactnavigation.org/

ReactNavigation的使用步骤：

①在当前项目下载必需的NPM包

npm i react-navigation

npm i react-native-gesture-handler

npm i react-native-reanimated

②创建路由词典 —— 称为“栈式路由导航器”

//App.js

let routes = createStackNavigator({

'login': LoginScreen, //排名第一的组件就是默认APP的首屏内容

'main': MainScreen,

})

③创建根组件，注册路由词典

//App.js

export default createAppContainer( routes )

④访问测试

摇一摇 > 重新加载即可

提示：RN应用完全不涉及浏览器，所以无法在任何地址栏中输入路由地址！只能通过ReactNavigation提供的路由跳转机制来跳转

课后任务：

①根据“RN阶段项目设计图”完成如下四屏组件的静态布局内容：

LoginScreen、MainScreen、ProductListScreen、ProductDetailScreen

注意：页头和标题栏由导航组件自动提供，无需手工编写

②阅读ReactNavigation手册，掌握页面跳转及页面传参的相关实现代码

|  |
| --- |
| 前端三大难点：  ①浏览器/设备兼容性  ②性能优化 —— 列表组件的性能优化问题？  ③Web安全 |